ALL YOU NEVER WANTED TO KNOW ABOUT THE BERRY III ENVIRONMENT

Christian Balkenius

Lund University Cognitive Science
Kungshuset, Lundagård
S–223 50 LUND, Sweden
E-mail: christian.balkenius@fil.lu.se

ABSTRACT: BERRY III is a simulator for autonomous robots and artificial creatures. This report describes the various elements of the simulated environment and the implementation of the physical part of the simulator. The body, the sensor and motor systems of the simulated creatures are described in detail together with the algorithms used to model interaction between objects. These algorithms include calculation of smell diffusion, tactile and visual input, movement of the creature and collision detection.

1. Introduction

BERRY III is a program that simulates neural network controlled artificial creatures, their environment and evolution. This report describes the details of the simulated environment that may be of interest to anyone wanting to replicate various experiments performed with the system and to anyone intending to write a similar simulation package.

The neural network that serves as the controller in the creatures and the implementation of simulated evolution are *not* described in this report. A general description of the simulated creatures can be found in Balkenius (1993a). For an overview of the implementation of the genetic algorithm used, see Balkenius (1993b).

2. Units and Measurements

Although BERRY is not a physical simulator, it is useful to define a number of units that are used to specify the measures of the components of the simulated environment. Explicit specification of measures may also be useful if BERRY is ever implement in a physical robot.

In the BERRY world, length is measured in LU (length units). This relieves us from specifying the *actual* size of objects and creatures in the environment. In the presentation on screen, 1 LU is arbitrarily represented by the width of one pixel. However, all calculations are made with full precision and all positions are specified with real values. Optionally, a grid may be used to constrain the locations of objects and the creatures.

Time is measured in Ticks. Since time in the BERRY world is discrete, a Tick is the time consumed by one iteration of all calculations in the system. Consequently, the length of a Tick is not fixed in relation to real time.

Below, I will also make use of some derived units in the measurement of velocity (LU/Tick) and for the specification of motor speed (rotations per Tick or RPT).

3. The Environment

The BERRY environment consists of an infinitely large two dimensional plane, though in practice, the environment is constrained by walls. By default, these limit the environment to a square with sides of 300 LU. The walls have the important property that they keep the creatures within their window on screen (However, some escapes has been known to occur at an early stage of system development).

Creatures and other objects are placed in this plane and are all simulated as two dimensional shapes such as circles and squares. Thus, the world is very similar to Flatland (cf. Abbott 1884/1991).

Points in the environment are referred to as a coordinate pair $p=(x, y)$. The coordinate system is mapped on the screen as shown in figure 1.

LUCS Minor 2, 1994. ISSN 1104–1609.

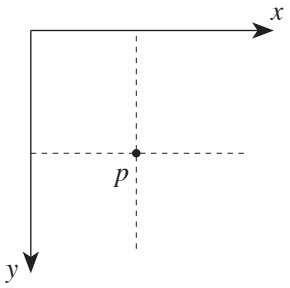

Figure 1. The Coordinate System of the Environment

4. Elements of the Environment

The BERRY environment consists of a number of objects such as walls, food and creatures. The goal in constructing the BERRY environment has not been to make it as realistic or as complex as possible. While very complex environments can give us insights that are not possible in a simple environment (cf. Tyrell, 1993), it is also very often important to strip the environment of all but the essential features. This is the approach taken in the present simulator.

4.1 WALLS

Walls are represented as lines in the plane. They are defined by the coordinates of its endpoints. Walls can be either opaque or non-opaque. Opaque walls do not let smells through while non-opaque walls do. The primary function of walls are to structure the environment into something a little more interesting than an empty surface.

A wall also has a colour and a height. Colour is defined as a point in RGB-space (i. e. red, green, blue intensities). Height is specified in LU and is typically 10. Height is admittedly a rather obscure notion in a two-dimensional environment. As we will see in section 7.4, it does have some uses however. Walls also have a number of graphical attributes such as luminance and reflectance but we will not consider them here.

4.2 SMELLY THINGS

All objects in the BERRY environment, except for walls, smell. An odour is defined as an eight component vector of smell intensities $o = <o_1, o_2, \ldots, o_8>$. These values are typically in the range 0..1 and describe a smell patterns. An intensity, I, is also associated with each smelly object. The smell leaving the object is given by $I \cdot o$. Refer to section 7.2 for a presentation of smell diffusion in the environment.

4.3 FOOD

There are four types of food in the environment: A, B, C and D. Food is modelled as circles with varying radius. Typically the radius is 0 in which case the food resides at a point. A food object may have a varying amount of food substance and a smell. When a creature eats, a certain amount, $b=<b_A, b_B, b_C, b_D>$, of food is transferred from the food object to the creature. When this happens, the food substance in the food object may optionally decrease with the same amount. It is also possible for the food substance to be regenerated at a fixed rate $r=<r_A, r_B, r_C, r_D>$. The smell intensity of the food object may also be correlated with the amount of food substance available.

4.4 AVERSIVE OBJECTS

An aversive object gives a creature an ‘electric shock’ if it comes into contact with it. These objects are circular or points, just like food, and also give off smells. When a creature is in contact with an aversive object, its shock sensor reacts.

4.5 GADGETS

A gadget is an object with special properties like a door with a lock. These object are used to make the environment more complex or tricky. The currently available gadgets are described below. In the future, other types of gadgets may also be included in the simulator.

4.5.1 Teleporters

Teleporters may not be one of the most essential object in a world but they do have a number of useful properties. Essentially, a teleporter is a gadget that consist of an entry area in the form of a circle and an exit point. If a creature makes contact with the entry area it is immediately teleported (i. e. moved) to the exit point. There are many uses for a mechanism like this. For instance, if a teleporter is placed in the goal box of a maze, the creature can be automatically transported to the start box for yet another trial. Teleporters can also be used to build mazes with topologies of arbitrary complexity. Like most things in the environment, teleporters smell.

4.5.2 Locked Doors

Locked Doors can be used to make the environment more interesting. This gadget consists of two separate objects, a door that can be temporarily opened and a circular region. When a creature makes contact with the circular region, the door is opened for a short period of time. Like teleporters, doors may be used to construct environments with non-reversible actions.

4.5.3 Distal Landmarks

A distal landmark is a special type of thing the location of which can be recognized from all positions in the environment. Landmarks are used by some creatures as guidance for spatial orientation.

4.6 AREAS

An area is a rectangular region of the plane with some special property. For example, there are safe areas where creatures cannot be attacked by predators. The centre of an area may give off a smell signal. In future versions of the simulator, creatures will be able to detect whether or not they find themselves within a given region.

4.7 PREDATORS

There are two types of (optional) predators in the environment. The first is the ground predator that uses the same type of body as the BERRY creatures to move around. The second is the air predator that moves in straight lines over the environment and attacks creatures close to its path. A number of different strategies can be selected for the predators that can either be lethal or operate as a moving aversive object. Both types of predators may give off smells.

4.8 IRRELEVANT CREATURES

Irrelevant creatures look exactly like BERRY creatures but are much more stupid (to limit the computational power used to simulate them). The role of these creatures is generally to be in the way and to confuse the real BERRY creatures. They make sure the environment is changing all the time.

5. THE BERRY CREATURE

We now turn to the specification of the BERRY creature as it looks in the current implementation. We will describe all parts of the creature except its nervous system.

5.1 THE BODY

The BERRY creature has a perfectly circular two dimensional body. The diameter of the body, d, is 6 LU (figure 2).

The creature has one whisker on each side of its body that are directed forwards. The whiskers have their origin at the centre of the body and have a variable length, s, typically 5 LU. The angles, φ, between the forward direction and the whiskers are variable but typically 45°. It is always the same for both whiskers.

A number of smell sensors are located at the end of the whiskers. In the current version of BERRY there are receptors for 8 different smells. The smell sensors are sensitive to the concentration of 8 different chemical substances called 1, 2,…, 8. Also, the body gives off a specific smell, o, that may be distinct for each individual creature.

There is *one* cyclopean eye in the front that is fixed in relation to the body. To look around, the creature must turn its entire body.

The body moves through the environment using two motors that drive wheels on each side of the body. By varying the speeds and directions of the motors, the creature can move forwards and backwards, and turn while moving or spin. The motor system is described further below.

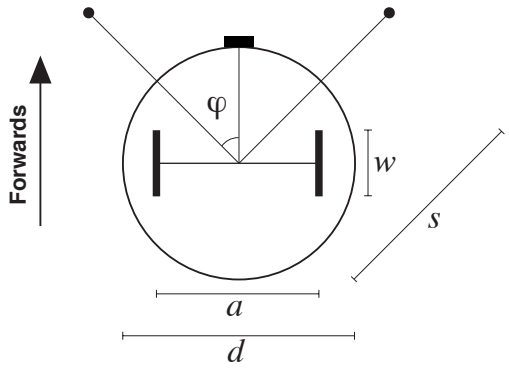

Figure 2. The body of a creature. By default a=2, w=2, d=6, s=5 and $\varphi=\pi/4$.

This type of general body architecture has been used in a number of studies with both physical robots and in simulated environments, most notably Braitenberg (1984). In these studies, the body and motor control is not the primary object of study. However, the type of output signals necessary to control the movement of the creature can easily be adapted to more developed models of locomotion (e.g. Beer 1990, 1992) or to real robots (cf. Hirose 1993, Mills 1993).

In summary, a creature is specified by the tuple $C=<d, \varphi, s, a, w, o>$, where d, φ, s, a, w, o are defined as in figure 2. The position of a creature in the environment is given by the tuple $<p_C, d_C>$, where p_C is the point at the centre of the creature and d_C is the direction of the creature specified as a vector of unit length (See figure 3).

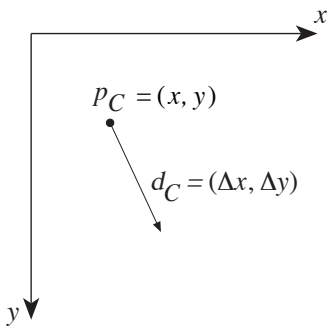

Figure 3. The specification of the place and the direction of the creature *C*.

5.2 FOOD DETECTORS AND REINFORCEMENT SIGNALS

There is one food detector for each of the four food types in the environment. When the creature is in contact with food, one of these detectors will react. It is possible for the creature to eat a food object only when the food detector reacts to it. Apart from generating eating, the food detectors can be used as signals that initiate reinforcement learning.

The body is also susceptible to 'electric shock', various aversive objects generate these signals if the body is in contact with them. Like the food detectors, the shock sensor can be used to drive learning. Collision with a wall may also activate the shock sensor.

5.3 METABOLISM

When the creature eats food, an amount of food substance is transferred from the food object to the creature. The current metabolic state of the creature is given by the vector $m=<f_A, f_B, f_C, f_D>$, where f_A is the amount of consumed food substance *A* etc. This vector may optionally decrease either as a function of time or of body movement. The metabolic state is used to calculate the explicit fitness of a creature. This can be done in a number of ways but will not be described here.

6. INTERACTION BETWEEN OBJECTS

This section describes how collision and contact between objects are calculated. These calculations are all based on standard linear algebra. An introduction to the necessary 2D-geometry can be found in (Foley et al. 1990).

6.1 COLLISIONS BETWEEN TWO CIRCULAR OBJECTS

Let O_0 and O_1 be two circular objects at positions $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$ with the radii r_0 and r_1. The two objects have collided if

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2} \le r_0 + r_1. \quad (1)$$

6.2 COLLISIONS BETWEEN A CIRCULAR OBJECT AND A LINE

Let O be a circular object at positions $p = (x, y)$ with the radius r and let L be the line between $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$. The line L has collided with O if the smallest distance from p to L is less than r. This is calculated in two steps. First we try to trivially reject the intersection between the two lines by checking if the rectangle spanned by the two points p_0 and p_1 intersects with the circular object, if it does not, we know that there is no collision. If the rectangle and the circle intersect, we have to check the distance between the line and the centre of O to see if it is less than the radius of O. If it is, we have a collision.

We can trivially reject the line if one of the following conditions does not hold

$$\min(x_0, x_1) - r \le x \le \max(x_0, x_1) + r, \quad (2)$$

$$\min(y_0, y_1) - r \le y \le \max(y_0, y_1) + r. \quad (3)$$

If both these conditions are true we test the distance between L and O. We have a collision if

$$(x - x_0)^2 + (y - y_0)^2 - r^2 \le \frac{\left[(x - x_0)(x_0 - x_1) + (y - y_0)(y_0 - y_1)\right]^2}{(x_0 - x_1)^2 + (y_0 - y_1)^2}. \quad (4)$$

6.3 INTERSECTIONS BETWEEN TWO LINES

Let L_0 and L_1 be two lines and let each line L_i be described by its end-points $p_{i0}=(x_{i0}, y_{i0})$ and $p_{i1}=(x_{i1}, y_{i1})$. To test intersection of the two lines we first check if we can trivially reject the lines. Thus, if any of the following conditions hold, we know the lines do *not* intersect.

$$\max(x_{00}, x_{01}) < \min(x_{10}, x_{11}). \quad (5)$$

$$\max(x_{10}, x_{11}) < \min(x_{00}, x_{01}). \quad (6)$$

$$\max(y_{00}, y_{01}) < \min(y_{10}, y_{11}). \quad (7)$$

$$\max(y_{10}, y_{11}) < \min(y_{00}, y_{01}). \quad (8)$$

If none of the above conditions are true we go on to calculate the intersection points of the two lines. This is done in a straight forward manner by calculating the equations for each of the two lines of the form $y_0=k_0x_0+b_0$ and $y_1=k_1x_1+b_1$. Thus,

$$k_i = (y_{i1} - y_{i0}) / (x_{i1} - x_{i0}). \quad (9)$$

$$b_i = y_{i0} + k_0 x_{i0}, \quad (10)$$

with the exception that $k_i=\infty$ and $b_i=0$ if $(x_{i0}-x_{il})=0$. The intersection point is given by the equations,

$$x = (b_1 - b_0) / (k_0 - k_1), \quad (11)$$

$$y = k_0 x + b_0. \quad (12)$$

Unfortunately, we first have to check for a number of degenerate cases. First, are the two lines identical? If they are, they intersect everywhere and we are done. Second, is $k_0=k_1$? If this is true, the lines are parallel and can never intersect. Third, is $k_0=\infty$? If it is, we replace (11) and (12) by the following.

$$x = x_{00}, \quad (13)$$

$$y = k_1 x + b_1. \quad (14)$$

Fourth, is $k_1=\infty$? If so, we replace (11) and (12) by,

$$x = x_{10}, \quad (15)$$

$$y = k_0 x + b_0. \quad (16)$$

At this point, we finish by checking to see if the intersection point (x, y) lies on both of the lines in the obvious way. If any of the following conditions are met, the lines intersect:

$$x < \min(x_{00}, x_{01}), \quad (17)$$

$$x > \max(x_{00}, x_{01}), \quad (18)$$

$$x < \min(x_{10}, x_{11}), \quad (19)$$

$$x > \max(x_{10}, x_{11}), \quad (20)$$

$$y < \min(y_{00}, y_{01}), \quad (21)$$

$$y > \max(y_{00}, y_{01}), \quad (22)$$

$$y < \min(y_{10}, y_{11}), \quad (23)$$

$$y > \max(y_{10}, y_{11}). \quad (24)$$

Note that all these tests are necessary. It does not suffice with conditions (5)-(8) (see figure 4). This completes the calculation of intersection between lines. Below we will see how this algorithm can be extended to test for admissible paths in the environment.

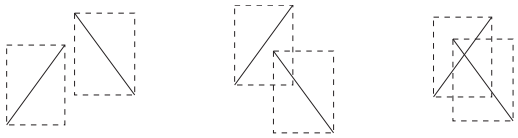

Figure 4. LEFT. Two lines that are trivially rejected by the test (5)-(8). MIDDLE. Lines that can not be trivially rejected but their intersection point lies outside one of the bounding rectangles. None of conditions (17)-(24) are met. RIGHT. Two lines that do intersect.

7. Calculation of Sensory Inputs

The BERRY creature has three types of sensory input, smell, touch and vision. These different types of sensors are described in this section.

7.1 Smell Sensors

The two symmetrically placed smell sensors are sensitive to eight 'chemical' substances called 1…8. The receptor for each smell r_i can be activated in the range [0, 1]. When $r_i=1$, it is not possible to activate the receptor further regardless of the smell intensity.

To calculate the activation of each receptor we first check to see what smelly things are within range and not behind any opaque wall. This is done in two steps. First, we test if the distance from the smelly thing to receptor is larger than the effective range of the smelly thing. If it is we are done with this smelly thing and go on to the next. Secondly, the algorithm described in section 6.3 is used on each wall in the environment and the line from the smell receptor to the smelly thing. If this line intersect with any wall, the receptor can not react to the smell. Otherwise we calculate the intensity, I, of the smell as described in the next section and add the product of this intensity and the smell level, s_i, to the receptor r_i. This procedure is iterated for each smelly thing in the environment and each smell receptor.

7.2 Smell Diffusion

By default, the intensity of a smell, $I(d)$, at the distance, d, from a smelly thing is given by,

$$I(d) = \left(\frac{1}{1 + cd^2}\right) i, \quad (25)$$

where c is a constant and i is the smell intensity at the centre of the object. It is also possible for the smell to disappear at a certain distance from the centre of the smelly thing. In this case the smell intensity is given by,

$$I(d) = \begin{cases} \left(\frac{1}{1+cd^2}\right) i & \text{if } d < \theta \\ 0 & \text{otherwise} \end{cases}. \quad (26)$$

It is very useful to be able to limit the range of the smell around an object in this way. If the smell sensors are outside this distance from a creature, we do not have to test for intersections with a wall or calculate the smell intensity. If (26) is used instead of (25) and there are many smelly things in the environment, the simulation speed is consequently substantially increased.

7.3 WHISKERS

The signals from the two whiskers are calculated by testing if the line corresponding to the whisker intersects with any object in the environment as described in section 6.2 and 6.3.

For reasons of complexity, the whiskers do not bend. This means that they may very well pass through objects. If it does, the whisker signal is set to 1, otherwise it is set to 0. This means that we do not get any information about how far into an object the whisker has reached. In most cases, this is not a problem, but it can make it hard for the creatures to get out when both whiskers react.

Sometimes when the whisker pass through a wall, the signals from the smell receptors are invalid at that moment. This problem can be fixed by inhibiting the smell receptors when the whiskers are in contact with an object.

7.4 EYE

The image at the cyclopean eye is constructed by building a viewer independent representation of the environment in the form of a BSP-tree (Foley et al. 1990). Once constructed, this environmental representation can be rendered very fast from any visual angle using the painters algorithm. Although the environment is mainly two-dimensional, objects are rendered with an additional dimension. This is where we make use of the heights of walls. The retinal image of the creatures are thus an ordinary two-dimensional projection of a three-dimensional model.

After each step, the BSP-tree is updated with the current position of all moving objects such as creatures. If there are many creatures at one time, this operation takes a large portion of the simulation time. In the future, we will make use of the property of BSP-trees that they constitute a Boolean algebra. Thus, instead of building a new BSP-tree each time the creatures move, it would be possible to keep one BSP-tree for each creature and one for the environment. By updating only the trees representing the creatures and then forming the conjunction of all trees, some computational cost would probably be gained. This ought to be the case, at least, when there are many objects in the static environment tree and not too many creatures.

7.5 SENSORY IMPERFECTION AND NOISE

To make the sensors more realistic it is possible to add noise to them in two ways. The first type of noise, N, is independent of the signal while the second type of noise, M, is proportional to the signal level. If s is the original signal from a sensor, the noisy output, $\hat{s}$, is given by,

$$\hat{s} = N + (1 + M)s. \quad (27)$$

N and M are random variables with gaussian distribution in the range $[0, n]$ and $[-m, m]$. If the noisy signal is not within the specified range, it is truncated to the closest valid value. In later versions of the simulator, other distributions will also be possible.

8. MOTOR CONTROL

When constructing the motor system of the creatures, the goal has been to make it as realistic as possible within the bounds of an essentially two-dimensional world. As a result, the simulation of movement is fairly accurate.

8.1 MOVEMENT OF THE CREATURE

Starting with the two motor signals L and R to the motors on each side of the body specified in RPT, and the position (x, y) of the creature, the movement of the creature is calculated in the following way.

First the distance travelled by each wheel (in LU) is calculated as,

$$d_L = 2\pi wL, \quad (28)$$

$$d_R = 2\pi wR. \quad (29)$$

Here, w, is the wheel diameter as described in section 5.1. Then the shortest distance, M, to the resulting position in the environment is calculated:

$$M = w\left(\frac{d_L + d_R}{d_L - d_R}\right)\sin\left(\frac{d_L - d_R}{2w}\right). \quad (30)$$

We now check to see if the creature is moving in a straight line. If this is the case we get the new position of the creature (x', y') by,

$$\begin{cases} x' = x + M\Delta x \\ y' = y + M\Delta y \end{cases}. \quad (31)$$

The path from (x, y) to (x', y') is checked to see if it intersects with any wall as described below. If it does, a binary search is carried out to obtain the closest intersection point with a wall (see section 8.2). This point is substituted for the original (x', y').

If the speeds of the two motors are different, the creature will turn and we have to make a more complex calculation. First we let the creature turn in the direction of the shortest path to its new position (se figure 5).

$$\begin{cases} \Delta x := \Delta x \cos(\varphi / 2) - \Delta y \sin(\varphi / 2) \\ \Delta y := \Delta x \sin(\varphi / 2) + \Delta y \cos(\varphi / 2) \end{cases} \tag{32}$$

where := represents a Pascal-type assignment and,

$$\varphi = \left(\frac{d_L - d_R}{w} \right). \tag{33}$$

We now let the creature move in its current direction $(\Delta x, \Delta y)$. If it does not collide with any obstacle within the distance M, we simply move the creature to the new position given by equation (31) and then let it turn the resulting half angle. Thus, equation (32) is repeated once more.

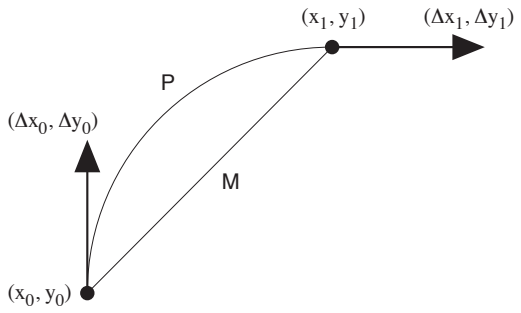

Figure 5. The correct path P and the approximate path M.

In the most complicated case when the creature both turns and collides with a wall, its new position is calculated in the following way. First the point of collision is calculated by binary search of the path (see section 8.2). The actual distance travelled by the creatures is d_I . We now calculate how much of its turning angle, φ', fits into this distance and what the resulting direction of the creature will be.

$$\varphi' = \frac{d_I \varphi}{M}, \tag{34}$$

$$\begin{cases} \Delta x' := \Delta x \cos \varphi' - \Delta y \sin \varphi' \\ \Delta y' := \Delta x \sin \varphi' + \Delta y \cos \varphi' \end{cases}, \tag{35}$$

$$\begin{cases} \Delta x = \Delta x' \Big/ \sqrt{(\Delta x')^2 + (\Delta y')^2} \\ \Delta y = \Delta y' \Big/ \sqrt{(\Delta x')^2 + (\Delta y')^2} \end{cases}. \tag{36}$$

The environment can optionally be modelled as having 'slippery walls'. In this case, φ' is set to φ in equation (34). The consequence is that the creature does not stick to a wall if it tries to move towards it. Instead it will slowly turn until it gets free.

Note that the path of the creature is approximated by a straight line. Consequently, the intersection point with an obstacle will be slightly offset from the correct path. However, if the speed of the creature is not too high, the error will be small.

8.2 CALCULATION OF ADMISSIBLE PATHS

Given a path P from the point $p_0=(x_0, y_0)$ to the point $p_1=(x_1, y_1)$, its admissibility is calculated as follows.

First we make some additions to the algorithm presented in section 6. For a path to be admissible, it is not sufficient that there be a free path from p_0 to p_1. It is also necessary for the creature to fit on all positions along this path. Since this would require a fairly complex calculation, we use a simple trick. We check only to see if the creature fits at position p_1 and not along the whole path. The shape of the region checked is shown in figure 6. This approximation gives nearly the correct result in most cases.

Some problems may occur if the creatures is moving very fast since it will then be able to pass through openings that are otherwise to small for it. A simple solution to this problem is to make all openings in the environment sufficiently large for the creature to pass through. In fact, this is never a limitation as long as all the creatures have the same size.

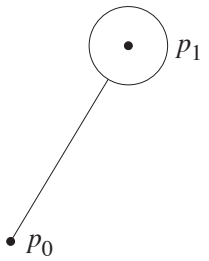

Figure 6. The path checked against walls and other obstacles. The large circle around p_1 has the same size as the creature.

Using the alterations of the algorithm described above, we check the path from p_0 to p_1. If there are no collisions, we are done. Otherwise we try to find the intersection of P that lies closest to p_0. This is done by a binary search of the path P.

Let $p'=(x', y')$ be the calculated point of collision and $p_m=(x_m, y_m)$ be the current search point. First we let p_0' be equal to p_0. We then calculate the current search point as:

$$\begin{cases} x_m := (x_0 + x_1)/2 \\ y_m := (y_0 + y_1)/2 \end{cases} \quad (37)$$

We then check if the path from p_0' to p_m intersects with any object. If it does, we set p_1 to p_m, otherwise we set p_0 to p_m. This procedure is repeated until p_0 and p_1 are sufficiently close to each other. When this is the case, p' is set to p_0.

Finally we calculate the distance from the start position to the intersection point, d_I, as

$$d_I = \sqrt{(x' - x_0')^2 + (y' - y_0')^2}\,. \quad (38)$$

8.3 MECHANICAL IMPERFECTION

In the current implementation, the only mechanical imperfection of the creature is a slight offset from the correct path when it collides with an obstacle. In future versions of the program, a mechanism for other variations of the mechanical system will be included.

9. THE MAIN TICK LOOP AND CREATURE UPDATE

The implementation of the simulator is made in a straightforward manner using one object for each thing in the environment. At each time increment a **Tick** message is sent to every object sequentially to let it update its state. As a result, the states of the objects are updated asynchronously. In most cases, this is the most realistic way to calculate the next state of the world but may cause some errors at times.

For example, if two creatures are moving fast and their paths intersect, it is possible for the creatures to pass straight through each other. Since one creature is moved at a time, they do not get a chance to collide. So far this has never been a problem sine the creatures usually do not move very fast. A more realistic updating mechanisms would be very time consuming and would not result in a very large advantage.

10. SYSTEM REQUIREMENTS, AVAILABILITY ETC.

BERRY III runs on Macintosh computers that have at least a 68020 processor, Color QuickDraw (but not necessarily colour) and System 7. The program is available directly from the author at no cost for non-commercial purposes or by anonymous ftp from LUCS.fil.lu.se in the directory:

/pub/simulators/BERRY–III.

The program is written using Symantec's THINK Pascal and makes heavy use of its object oriented features.

11. CONCLUSION

The main features of the BERRY simulator and the implementation of the creature body and its movement have been presented. An overview of the objects of the simulated environment has been given together with the implementation of interactions between objects.

REFERENCES

Abbott, E. A., (1884/1991), *Flatland*, Princeton Univerity Press, Princeton, NJ.

Balkenius, C., (1994), "Evolving artificial creatures", *in preparation*.

Balkenius, C., (1993), "Natural intelligence for autonomous agents". In B. Svensson (ed.) *Proceedings of the international workshop on mechatronical computer systems for perception and action*, Halmstad University.

Beer, D. B., (1990), *Intelligence as adaptive behavior*, Academic Press, San Diego.

Beer, R. A., Chiel, H. J., Quinn, R. D. & Larsson, P., (1992), "A distributed neural network architecture for hexapod robot locomotion", *Neural Computation*, **4**, 356–365.

Foley, J. D., van Dam, A., Feiner, S. K. & Hughes, J. F., (1990), *Computer Graphics*, Addison–Wesley, Readings, MA.

Hirose, S., (1993), *Bilogically inspired robots*, Oxford University Press, Oxford.

Mills, J. W., (1993), "Stiquito: A small, simple, inexpensive hexapod robot. Part 1. Locomotion and hard-wired control", Computer Science Department, Indiana University.

Tyrell, T., (1993), "Computational mechanisms for action selection", Ph. D. Thesis, University of Edinburgh.